QUANTUM ENTANGLEMENT, FLIPPING OUT AND INVERSE PROPERTIES

See Named Property Inverses: Yay or Nay for a shorter version of this paper

By: Michael Uschold

Property Inverses and Perspectives

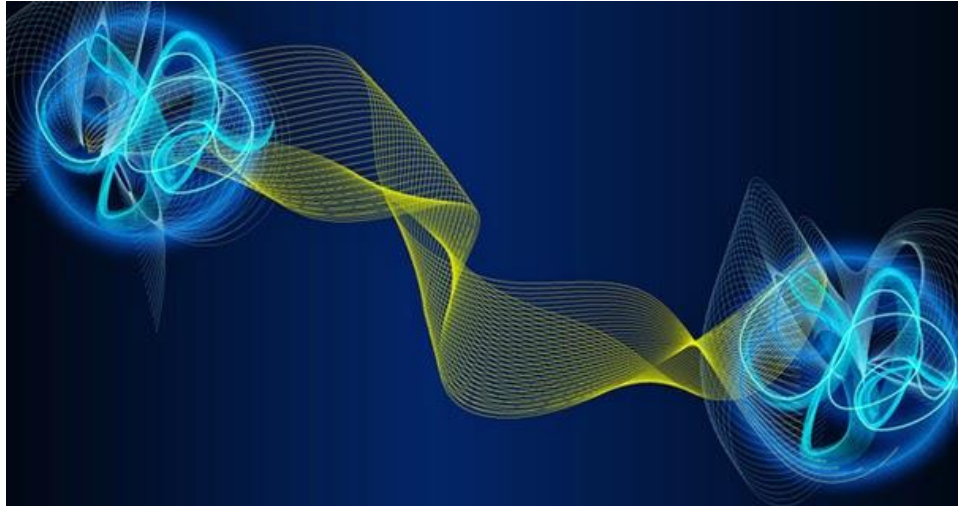

Figure 1: Quantum Entanglement

An OWL property represents a way that two things can be related to each other, e.g. being a parent or guaranteeing a loan. An OWL property is directional, which means it corresponds to the perspective of exactly one of the related things. For example, the child has a parent, but the parent has a child. The US government guarantees a loan, but the loan is guaranteed by the US government.

It is important to understand that logically, both perspectives always exist; they are joined at the hip. If Michael has Joan as a parent, then it is necessarily true that Joan has Michael as a child – and vice versa. If from one perspective, a new relationship link is created or an existing one is broken, then that change is immediately reflected when viewed from the other perspective. This is a bit like two quantumly entangled particles. The change in one is instantly reflected in the other, even if they are separated by millions of light years. Inverse properties and entangled particles are more like two sides of the same coin, than two different coins.

Figure 2: Two sides of the same coin.

In OWL we call the property that is from the other perspective the inverse property. Given that a property and its inverse are inseparable, technically, you cannot create or use one without [implicitly] creating or using the other. If you create a property hasParent, there is an OWL syntax that lets you refer to and use that property's inverse. In Manchester syntax you would write: "inverse(hasParent)". The term 'inverse' is a function that takes an object property as an argument and returns the inverse of that property. If you assert that Michael hasParent Joan, then the inverse assertion, Joan inverse(hasParent) Michael, is inferred to hold. If you decide to give the inverse property the name parentOf, then the inverse assertion is that Joan parentOf Michael. This is summarized in Figure 3 and the table below.

Subject	**Predicate**	**Object**
Michael	hasParent	Joan
Joan	parentOf	Michael
Joan	inverse(hasParent)	Michael

In the case of guaranteeing a loan, we might call the property from the perspective of the guarantor: 'guarantees'. We can use it to assert that `US_Govt :guarantees Loan123`. This implicitly asserts that `Loan123 inverse (:guarantees) US_Govt`. Again, we could decide to create an explicit inverse property. Can you think of a good name for it? It is good practice to choose terms that can result in triples that can be read in somewhat English-like expressions. Michael hasParent Joan. US_Govt guarantees Loan123. So what is the blank that would sound right here: Loan123 US_Govt. We will pick this up again below.

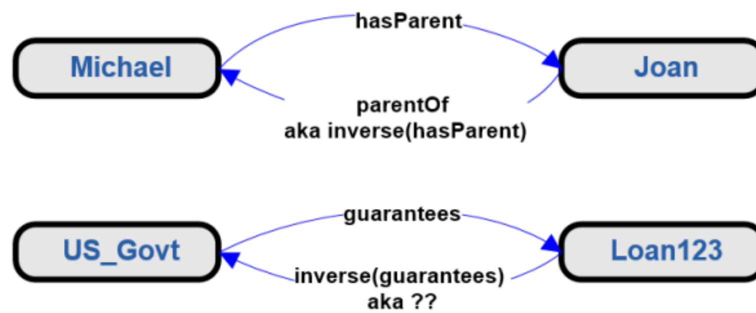

Figure 3: Properties with named and anonymous inverses.

Should you have a named inverse?

Here we consider the ramifications of having a named property inverses. There is no universal agreement on this issue, and at Semantic Arts, we have gone back and forth. Initially, we created them as a general rule, but then we noticed some down sides, so now we are more careful. Below are four downsides of using named inverses (roughly in order of growing importance). The first two relate to ease of learning and understanding the ontology. The last two relate inference and triple stores.

1. *Names*: It can be difficult to think of a good name for the inverse, so you might as well just use the syntax that explicitly says it is the inverse. It will likely be easier to understand.
2. *Cluttered property hierarchy*: Too many inverses can significantly clutter up the property hierarchy, making it difficult to find the property you need, and more generally, to learn and understand what properties there are in the ontology, and what they mean.

3. *Slower inference*: Too many named inverses can significantly slow down inference.
4. *More space*: If you run inference and materialize the triples, a named inverse will double the number of triples that use a given property.

Except for the second these are fairly self-explanatory, and it's a bit of a long explanation, so we will defer it to the end of the article.

Given these downsides, are there any situations when you still want to create named inverses? When might the downsides not be so bad, or simply fail to arise? When might the benefits be compelling enough to outweigh the downsides? The main benefit for named inverses is convenience and clarity. It is just easier and more natural to just say "guarantees" instead of "inverse(isGuaranteedBy)".

Names: We use gist:fromAgent and gist:toAgent to link an email message to the recipient and to the sender. What would you call this property from the recipient's perspective pointing back to the message? There is no obvious short, simple, and intuitive name that readily comes to mind. One possible name for the inverse is: messageSentToAgent. For another example, consider a product under development and there is a property linking it to the plant where they intend to manufacture it. So the property from the perspective of the product, might be called intendedManufacturingPlant. But what would you call it going the other direction? Sometimes there is no good answer. This may be the case when it is hardly ever used from the other perspective, so there is no common English term. You can make up a name like "intendedToManufacture", but it is not particularly satisfying. The meaning of inverse(intendedManufacturingPlant) might be equally or more clear, and none of the other downsides would apply.

Fortunately this is the exception, most of the time coming up with reasonable names for inverse properties is not so hard. So the first downside often does not arise.

Cluttered property hierarchy: There will be up to twice as many properties in the hierarchy, and the properties are not connected to their inverses in any way. With today's tools, there is no getting around having the inverses clutter up the property hierarchy. The only option is to use fewer inverses (unless you have a better idea and want to build your own plugin to view properties).

Slower inference: Inference engines are getting better and better, and for many modest-sized ontologies having inverses will not be a problem. If inference does start slowing down, it could be any number of things besides inverses. If it comes to that, then removing inverses is one of the things you can try to improve efficiency. Start with the ones that are not getting much use.

More Space: If you do not anticipate building a triple store based on the ontology and running inference and materializing the inferences, then that will also not be an issue.

Naming Patterns

Above we saw two examples of named inverses: hasParent/parentOf and guarantees/guaranteedBy. These exemplify two common patterns.

- hasX / xOf pattern – Other examples include: hasOwner / ownerOf, hasMember / MemberOf; there are many others.
- does / doneBy pattern: guarantees / guaranteedBy, identifies / identifiedBy, occupies / occupiedBy.

People prefer different linguistic styles, for example some people prefer a does / isDoneBy pattern. For example: isGuaranteedBy, isIdentifiedBy, isOccupiedBy. This works equally well.

Is There a Preferred Perspective?

Given a property and its inverse, is there a preferred perspective? If so, how can we identify it? For example, does it matter whether you represent the perspective of the child (hasParent) or of the parent (parentOf)? Should you prefer guarantees, or guaranteedBy? Should you prefer the perspective of the person (toAgent) or of the message (messageSentToAgent)? If you are not representing the inverse, the preferred perspective is the one you choose to represent. In the above examples, which perspective would you choose to represent if you only chose one?

What if you do explicitly define an inverse? Is there a preferred perspective then? Let's look at the following OWL in RDF/XML syntax for the hasParent(parentOf) example in a fictitious geneology namespace, gen.

```
<!- Define the property hasParent -->

   <owl:ObjectProperty rdf:about="&gen;hasParent"/>

<!- Define parentOf to be the inverse of hasParent -->

   <owl:ObjectProperty rdf:about="&gen;parentOf">

      <owl:inverseOf rdf:resource="&gen;hasParent"/>

   </owl:ObjectProperty>
```

Logically there is no preferred perspective, and any given ontology tool may or may not indicate a difference between the two perspectives. However, there is a difference evident in the OWL. One property has to already be defined in order for the inverse is defined in terms of it. So, you can think of the property that was defined 'first' to be the preferred perspective. In this case, it would be hasParent. So, to summarize:

- If you are not representing the inverse, you are preferring the perspective that you choose to represent.
- If you are representing the inverse property, and you want to prefer one perspective over the other, then the preferred perspective is the one that is used to define the inverse.

Logically, there is no preferred perspective, but there are practical considerations. You may be able to anticipate that target users of the ontology are likely to prefer one perspective. For example, in a messaging database or application, the user is almost always going to be 'on' a particular message looking at who is in the From or To fields. So you might prefer toAgent over messageSentToAgent. The other perspective can also be relevant, in this case search, where you wish to find all the message sent to or from a given individual (e.g. the sent folder).

If you are building a loans ontology for a loan company, users are probably thinking about the loan and wondering who guarantees it. It may be less likely that they will be thinking about a person or company and asking what loans they are guaranteeing. This would suggest that the perspective of the loan is preferred, and the property from that perspective would be called 'guaranteedBy'. This could matter from the

perspective of building out triples data. In addition, the properties may show up differently in the ontology editing and visualization tools, especially in inferences are not shown. This can impact ease of learning and understanding the ontology. Note that this is highly subjective, and not everyone will have the same preferences.

Conclusion

There are always tradeoffs. It can still make sense to use named inverses if there are good names for inverses, and if both the property and its inverse are frequently used in the ontology, and also in triples that are based on the ontology. Here is our advice:

1. Be selective in creating named inverses.
2. Only use them when they will get frequent use, thereby justifying their existence.
3. Expected scenarios that are important to target users should determine which perspective you pick for representing an important relationship.
4. Logically, there is not a preferred perspective for a property vs. its inverse, but there may be practical reasons to prefer one over the other.

Appendix: Cluttering up the Property Hierarchy

To understand how too many inverses can clutter up the property hierarchy, say we have four relationships that we care about (this example is a bit artificial, but makes the point).

1. hasChild
2. isGuaranteedBy
3. likes
4. rides

If we create these properties in say Protégé or TopBraid Composer, we see an alphabetical list like above. If we add in the inverses, look what happens:

1. childOf
2. guarantees
3. hasChild
4. isGuaranteedBy
5. isLikedBy
6. isRiddenBy
7. likes
8. rides

This is a very short list; in a reasonable-sized ontology there are dozens of properties that need to be scrolled through. The inverses are often nowhere near each other. If I have just spent three minutes studying and understanding the property: "guarantees", I don't want or need to look at the inverse – yet there it is, in my face, taking up screen real estate. If you create named inverses for all the properties, there are twice as many properties. This example shows a flat set of properties. The problem is much worse when there is a property hierarchy. One property may be two or three levels down in the hierarchy, but its inverse is at the top level. This is confusing, especially for beginners. Advanced users will get why that is, but those named inverses still get in the way.

With today's tools, there is no getting around the cluttering of the property hierarchy if you create a lot of inverses. Why? Because no tool vendor has provided a way to display a property hierarchy including inverses in a concise easy to read manner. But it's not rocket science, it's quite simple. Just keep the property and its inverse on the same line. After all they are not so much different coins, as they are two sides of the same coin. It can look like this:

1. hasChild (childOf)
2. isGuaranteedBy (guarantees)
3. likes (isLikedBy)
4. rides (isRiddenBy)

Voila, four relationships and four properties. It is much easier and faster to see what is going on. They will show up nicely in an indented property hierarchy. I wish the tool vendors would do something like this. There is of course a question of how to choose which property is in parentheses, but even an arbitrary choice would probably be a big improvement.

11 Old Town Square
Suite 200
Fort Collins, CO 80524

970-490-2224
305-425-2224
info@semanticarts.com

© Semantic Arts, Inc.